Equal Time for Data on the Internet with WebSemantics

George A. Mihaila, Louiqa Raschid, Anthony Tomasic

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Equal Time for Data on the Internet with WebSemantics

George A. Mihaila, Louiqa Raschid, et Anthony Tomasic

N ° 3136

Mars 1997

THÈME 1

ISSN 0249-6399

INRIA
ROCQUENCOURT

Equal Time for Data on the Internet with WebSemantics

George A. Mihaila*, Louiqa Raschid†, et Anthony Tomasic‡

Thème 1 — Réseaux et systèmes
Projet Rodin

Rapport de recherche n ° 3136 — Mars 1997 — 20 pages

Abstract: The Internet contains a large amount of structured data, accessible via ftp files, ODBC connections, or embedded in HTML documents. This data cannot be effectively used for several reasons. First, the structure of the data (type, format, etc.), is usually not described. Second, no tools exist for locating sources for structured data. Third, accessing the data requires either finding or building translators, and handling multiple incompatible protocols. This paper describes a system, WebSemantics, that provides integrated access to tools for accomplishing these tasks. We describe a protocol and architecture for locating data sources and translators; a query language and query processing system for accessing data; and a common protocol for data exchange. The result is a system that gives "equal-time" to data access on the Internet. Our goal is to support a world-wide community of users who share data with the same economy and ease with which documents are currently shared.

Key-words: Internet, Heterogeneous Database, Wrapper, Mediator, Translator, World-Wide Web

(Résumé : tsvp)

* University of Toronto Department of Computer Science. E-mail: georgem@cs.toronto.edu
† University of Maryland Department of Computer Science. E-mail: louiqa@umiacs.umd.edu
‡ E-mail: Anthony.Tomasic@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : (33) 01 39 63 55 11 – Télécopie : (33) 01 39 63 53 30

"Equal time" pour information en l'Internet avec WebSemantics

Résumé : L'Internet contient une grande quantité de données structurées, accessibles par des fichiers ftp, des connections ODBC, ou contenues dans des documents HTML. Ces données ne peuvent pas être utilisées d'une manière efficace à cause de plusieurs raisons. Premièrement, la structure de ces données (le type, le format, etc.), n'est pas décrite, normalement. Deuxièmement, il n'y a pas d'outils pour localiser des sources pour des données structurées. Troisièmement, pour accéder les données on doit soit trouver soit développer des traducteurs pour plusieurs protocoles incompatibles. Ce papier décrit un système, WebSemantics, qui fournit un accès intégré aux outils pour achever ces tâches. On décrit un protocole et une architecture pour localiser les sources de données et les traducteurs; un langage de requête et un système d'exécution des requêtes pour accéder les données; et un protocole commun pour l'échange des données. Le résultat est un système qui donne "equal time" pour l'accès des données sur l'Internet. Notre but est d'aider une communauté globale d'utilisateurs à partager des données d'une manière aussi simple que la façon dont on partage maintenant les documents.

Mots-clé : Internet, Base de donnée, Médiateur, Adapter, Traducteur, World-Wide Web

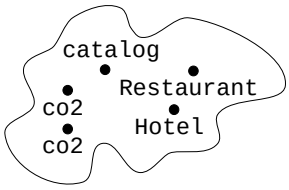

Figure 1: The space of data sources (represented by a circle) and the associated type.

1 Introduction

The amount of *structured* data available today in the INTERNET makes it an invaluable resource of information. By structured data, we mean data associated with a well defined *type*, e.g., tables of data located in `ftp`-able files. Effective use of this data is difficult due to the absence of various services in the INTERNET. For structured data, there are no well defined ways for the following:

- *describing* the structure and semantics of the data;
- *locating* relevant data; or
- *accessing* data in an efficient and uniform manner.

Consider a file containing a table of environmental measurements, e.g., the amount of CO_2 in the air at Notre Dame, in Paris, on February 9th, 1997, measured in one hour intervals. There is no agreement upon data model for describing the structure and semantics of the types in a file. Each file may have a different data format, and the type of the information in the file is either undocumented or documented only in human-readable form, such as a text document, and not in machine-readable form.

Locating relevant data is exceedingly difficult. For an environmental scientist to locate the file, she must find the `ftp` address, directory path, and file name of the file. There is no widely accepted agreement for locating this information. Users find this information in various *ad hoc* ways: from other users, news groups, conferences, advertisements, etc. In addition, none of these methods are ideal for finding information of a particular type.

Finally, access to the data is not efficient or homogeneous. Even for data with a well-understood data format, type and semantics, the data must be down-loaded, (usually by hand), and pre-processed, (usually with a FORTRAN program), before the data can be used by the environmental scientist. In addition, the data may be available via some other protocol besides `ftp`.

A second major source of structured information on the INTERNET is from databases. Access to a database is generally achieved through a gateway protocol such as ODBC. While access is efficient, this method still suffers from the above problems. Support for a data dictionary is cumbersome in ODBC; several queries need to be processed to obtain the types. Further, the same type in different databases could conflict with each other, since there is no standard type system. No mechanism is provided for the location of ODBC server databases. There is also the drawback that the protocol for accessing data from a database and from `ftp` files is completely different.

Besides structured data, the INTERNET contains a vast amount of semi-structured data, mostly embedded in HTML files. Data stored in semi-structured HTML files suffers from the same limitations of location and access. If the HTML file contains a human-readable description of the meaning of the data, in addition to the data itself, then locating data stored this way is somewhat easier since the WWW can be searched using information retrieval techniques. However, the lack of a formal type makes it difficult to access the kind of data.

In summary, although much interesting structured data, and semi-structured data, is available on the INTERNET, this data is difficult to *describe*, *locate*, or *access*. Users or application programs have to deal with heterogeneity of access protocols and data formats. In this paper we describe a system, call WEBSEMANTICS, that attacks these problems.

1.1 WEBSEMANTICS Support for Structured Data

For structured data, WEBSEMANTICS first provides a data model and tools, for users to describe a "cloud" of types, as illustrated in Figure 1. By *type*, we mean a user-defined class or interface in the WEBSEMANTICS data model. Each circle represents a data source which has an INTERNET address and exports a type as indicated

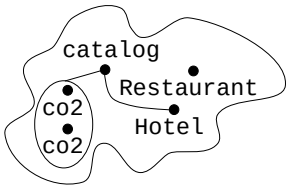

Figure 2: A catalog server knows the address of the data sources for type co2.

by the type name near the circle. WEBSEMANTICS extends the protocol of the mediator/wrapper architecture of DISCO [TRV96] for accessing heterogeneous data sources. More detail on the architecture and protocols of WEBSEMANTICS is given in Section 4.

To access data in a homogeneous way among a heterogeneous collection of access protocols and data formats, WEBSEMANTICS uses *translators*. (A translator is an extension to the *wrapper* concept in heterogeneous distributed databases.) Translators are services that provide gateways between the WEBSEMANTICS system and a data source. Translators accept (sub)queries from WEBSEMANTICS, process the queries in a way dependent on the data source, and return answers to WEBSEMANTICS. This communication is accomplished using the WEBSEMANTICS sub-query protocol (WS-SQP). Translators are described in detail in Section 7.

WEBSEMANTICS provides a novel *catalog server* to aid in the location of translators. For instance, the catalog service can, in response to a query, efficiently provide the addresses of all data sources that contain data of type `co2` and of type `Hotel`, as illustrated in Figure 2. The WEBSEMANTICS system accesses data by first accessing the catalog, finding the catalog entry for the some data, and then accessing the data by using the catalog entry. In addition to locating data sources, the catalog server aids in the integration of data, since it can be used as a source of standardized types to be shared among multiple users. We do not require that every data source be recorded in a catalog server.

WEBSEMANTICS aids in the homogeneous and efficient access to data by providing a query language and query processing system that integrates the location of data sources and the access to data sources. The access to data sources is through a heterogeneous collection of protocols.

For example, consider the environmental scientist who produced the CO_2 data mentioned above as the result of scientific measurements. The data is stored at an `ftp` file at the address `ftp://a.b.c/dir/data` and is formatted as a table. She has published a report in a journal that reports on the data and its interpretation. A second scientist has read the report and would like to use the data. Using existing technology, he is forced to copy the data locally via `ftp` and then determine the schema of the data, usually via e-mail with the first scientist. In WEBSEMANTICS, this exchange of information is accomplished in a simple way. The first scientist *registers* a way to access the data with a catalog server. Registration means transmitting a *triple* of information to the catalog server consisting of the address of the data, the type of the data, and the name of the translator of the data. Suppose our first scientist has a translator `ftp-table-translator` that translates a `ftp` file in a given format. Then the scientist would transmit the following triple

```
(ftp://a.b.c/dir/data,
 interface co2 {
    attribute Integer time;
    attribute Integer date;
    attribute Integer location;
    attribute Integer value },
 ftp-table-translator)
```

as an update to the catalog server. We call these triples *catalog entries*.[1]

Once the catalog entry for the data has been registered, the first scientist includes the address a of the catalog server in the journal article. To access the data, the second scientist down-loads the WEBSEMANTICS system into his browser, attaches the catalog server to the system by specifying the a, and accesses the data with the following query:

```
select x.time, x.date, x.location, x.value
from co2 x
```

[1] Technically, the WEBSEMANTICS system can function without catalog servers. However, this requires the user to specify catalog entries directly in a query to specify the location of some data.

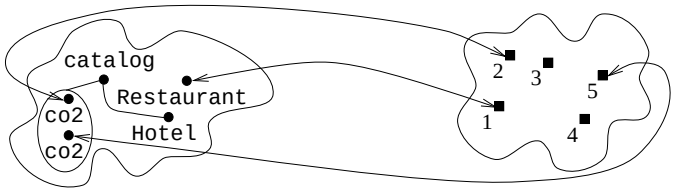

Figure 3: Links between the WEBSEMANTICS world and the WWW.

The WEBSEMANTICS system will access the data by first accessing the catalog at address a, finding the catalog entry for the `co2` data, and then accessing the data by using the catalog entry. Thus, at this point, the scientist is free to browse or down-load the data with the full functionality of the WEBSEMANTICS query language. In addition, should he have addition data of the `co2`, he can export his data into the WEBSEMANTICS system by adding a catalog entry to the same catalog server.

In our example, we assumed that the meaning of the `co2` type is understood to be a standard type, by both the data provider, (the first scientist), and the user of the data, (the second scientist). This assumes a degree of *semantic homogeneity* of the types; we do not need to impose homogeneity of the access protocols. In this example, this assumption of standard types is reasonable. First, the scientific document itself can clear up any misunderstandings. Second, there are national standards for environmental information. For example, the Sandre French national standard for environmental information [San95] consists of hundreds of relations and associated documents that precisely define the measurement of some types of environmental data. In fact, any community of users can create a new standard for types by simply providing a catalog server for the types.

There are situations however, where structured data does not support this assumption of semantic homogeneity. For example, two `employee` types from two databases, accessed via ODBC, may not be identical. There has been much research on semantic heterogeneity [Ken89, Kim95]); it is a complex problem and we expect that this research will continue. However, many application areas have defined sophisticated standard semantics. These semantics are described in (lengthy) documents that define the meaning of each field and provide a record format. For instance, US MARC [USM96] is a standard that defines bibliography records for libraries – it includes definitions of author, title, editor, etc. in detail. The STARTS project [LGP97] defines standard meta-data semantics for WWW search engines. There is an initiative to harmonize metadata standards and to develop interoperable metadata registries accessible via the WWW [WMR97]. While all of these initiatives propose standards for data formats and semantics, they do *not* generally define protocols for the location of data sources and translators, nor do they provide access to the associated data. These latter tasks are the focus of WEBSEMANTICS.

1.2 WEBSEMANTICS and the World Wide Web

The above description is entirely independent of the WWW, with the exception of the issue of semi-structured data. We view the WEBSEMANTICS data space as complementary to the WWW. As described above, WEBSEMANTICS requires users to "publish" in catalog servers the address of data in the form of catalog entries. Because of the ease of use and broad popularity of the WWW, it is advantageous to provide a way to map WEBSEMANTICS catalog entries into WWW documents, such that publishing of an HTML document simultaneously publishes a catalog entry and information about the catalog entry. Thus, users browsing the WWW can read the description of some data source and automatically have query access to the source.

To accomplish this mapping, *catalog entries are paired with HTML documents.* (These pairs of text and type are called *dyads* in [TS97].) These documents can be used in lieu of a catalog entry. That is, a triple does not need to be registered with a WEBSEMANTICS catalog server and we provide a mechanism, based on WWW information retrieval, to locate the catalog entry and the associated data. Figure 3 shows this relationship between WEBSEMANTICS and the WWW. Each arrow indicates a reference between an HTML document containing a catalog entry, on the right, and the corresponding data source, on the left. Thus, we have two sets of objects and a relation between them. The first consequence of this mapping is improved flexibility in access since a user simple needs an HTML address to access data. Secondly, the user has better access to various subsets of data sources and HTML pages. Subsets of data sources in the WEBSEMANTICS world and subsets of HTML pages in the WWW world can be easily constructed. In the WEBSEMANTICS world this is accomplished via the catalog server. In the WWW world this is accomplished via informational retrieval search

engines. In addition, the WEBSEMANTICS language permits access to the *image* of a given subset in the other world. For instance, the image of the subset of Figure 2 is documents 2 and 5. Third, users no longer need to explicitly catalog entries in a catalog, since the catalog can simply sweep the WWW looking for catalog entries.

Consider the example of an administrator who is organizing a lunch meeting in Paris for people arriving from other countries. The administrator must find both a restaurant and a hotel for the meeting. Thus, the administrator is interested in a data source that exports restaurant reviews and a data source that exports hotel reviews. The administrator examines a standard WWW site that lists the standard types available and finds two of interest: the type `Restaurant` and the type `Hotel`. The administrator would like a table of all excellent restaurants (two or three star) as rated by the guide *Gault Millau* and excellent hotels (four or five star) that are in the same *arrondissement*. The following query expresses this table:

```
select (y.name, b.name)
from x mentions ``Gault Millau'', Restaurant y in x,
     b exports type Hotel,
where y.arrondissement = b.arrondissement and y.stars > 1 and b.stars > 3
```

The `select` and `where` parts have a classical interpretation. The `from` clause has the following meaning. The variable `x` is bound to all URLs that match (in the information retrieval sense) the phrase "Gault Millau". The variable `y` is bound to any data source, identified in URL `x`, that provides data of the `Restaurant` type. If no such type can be identified by the URL, then it is ignored. The variable `b` is bound to any data source that matches (in the sense of *type matching*) the `Hotel` type. Both the processing of `mentions` and `exports type` are accomplished by consulting indexes. In the former case, an information retrieval index of the WWW is consulted. In the latter case, we provide an index for the catalog server that does type matching. In our example, we again assume a level of semantic homogeneity; i.e., the attribute `arrondissement` of the `Restaurant` and `Hotel` types can be sensibly compared using equality.

Another contribution of WEBSEMANTICS permits data to be located in HTML documents via a WEBSEMANTICS *data exchange format* (WS-DEF) and a special translator for this data format. This permits, as a special case, the processing of queries over data that directly resides in HTML documents. However, we note that this is a particular application of the WEBSEMANTICS translator protocol to the WWW. The existence of data in HTML documents is not central to the WEBSEMANTICS architecture. This special case of translators for processing WEBSEMANTICS data in HTML documents is described in Section 7.

1.3 Summary

In summary, WEBSEMANTICS addresses the problem of describing data by providing a data model, locating data by providing catalog servers, and accessing structured (and some semi-structured) data by providing a translator system and a query processor. The system operations in an environment with a heterogeneous collection of access protocols and data formats. For the problem of describing data, (a) we define an architecture based on translator technology that permits dynamic connection to translators; and (b) we define a strongly typed data model. For the problem of locating data, (a) we define a catalog service (that performs type matching); and (b) we define a language for declaring translators and types in an HTML based syntax. For the problem of accessing data, (a) we define a query language that combines location of data with the access of data; (b) we define an algebra for processing queries in the query language; (c) and we define a protocol for data access from translators. As a special case, we consider a special case of the translator when data is contained in HTML documents. The result is a system that gives "equal-time"[2] for data access on the INTERNET. Our goal is to support a world-wide community of users to share data with the same economy and ease with which documents are currently shared.

The paper is organized as follows. Section 2 describes the WEBSEMANTICS data model. Section 3 describes the catalog service. Section 4 describes the architecture. Section 5 describes the query language, its meaning, and the algebra for executing queries. Section 7 describes the interface to translators. Section 8 discusses related work, and Section 9 concludes the paper by summarizing its contributions and describes the status of our prototype implementation.

[2]This expression also means that television broadcasting time for the discussion of an issue is equally divided among different political parties.

2 Data Model

2.1 Overview

WEBSEMANTICS currently supports a very simple object data model. (The model is similar to the *Level 1 Mediator Standard* proposed in [BRU97].) The model is strongly typed and consists of base types and user defined types (classes). A type defines a set of properties, which are either attributes or relationships. An attribute is defined over basic types. A (binary) relationship is defined between two object types and has an identifying label representing a path traversal between the two types. We support the basic arithmetic comparison operators and boolean operations. The BNF description of the syntax for the data model is described in Appendix A. The syntax is similar to the ODMG object data model, proposed by the ODMG committee [C+96]. A simple example of user defined type is:

```
interface Restaurant {
  attribute String name;
  relationship Business headquarters;
}
```

The data model is simple and does not support many features (e.g., type hierarchies, structured objects, etc.) nor does it support global object identity. In Section 7, we consider the impact of the lack of global object identity.

2.2 Data Exchange Format (WS-DEF)

WEBSEMANTICS also provides support for processing queries directly over data that resides in HTML documents. Suppose the document contains objects of type `Restaurant`. In WEBSEMANTICS this data can be simply expressed as follows:

```
<HTML>
<HEAD>
<WebSemantics>

interface Restaurant {
  attribute String name;
  attribute String street;
  attribute Integer number;
  attribute Integer arrondissement;
  attribute Integer price;
  attribute Integer stars;
  relationship Business headquarters;
}

Restaurant("Caveau Francois Villon","rue de l'Arbre Sec", 64, 3, 230, 0,
"http://server.com/business.html#villon")

Restaurant("Chez Allard", "rue de l'Eperon", 1, 6, 370, 0,
"http://server.com/business.html#allard")

Restaurant("Faugeron", "rue de Longchamp", 52, 16, 580, 0,
"http://server.com/business.html#faugeron")
```

etc.

```
</WebSemantics>
<HEAD>
<BODY>
...
</BODY>
</HTML>
```

As the example shows, relationships are encoded using object references. We assume that the document "http://server.com/business.html" contains `Business` objects. Individual objects in this document are identified by HTML labels (`<A NAME=label>`) placed just before the object.

3 Catalog Server

For each data source willing to provide access to data through WEBSEMANTICS, a catalog entry must be created. Catalog entries can be added directly to a catalog, or they can reside in documents. These documents contain two parts: (i) the catalog entry (in a suitable syntax) and (ii) a textual description of the data present in the data source. For example, to describe a database with tourist information for Paris, one would write an HTML document like the one below:

```
<HTML>
<HEAD>
<WebSemantics>
interface Restaurant {
  attribute String name;
  attribute String street;

etc., including the definition of Hotel

}

CatalogEntry(Restaurant, rmi://server/transR, http://www.paris.guide)
CatalogEntry(Hotel, rmi://server/transH, http://www.michelin)
</WebSemantics>
</HEAD>
<BODY>
Welcome to the "Gault Milau" guide online. We provide information
about the best restaurants and hotels in Paris.

etc.

</BODY>
</HTML>
```

The data source connection information is written between special `<WebSemantics>` HTML tags so it will be ignored by a regular browser. [3]

After a new WEBSEMANTICS document is created, its author could register it with a catalog server by using a form based interface to the server. We also envision using robot technology to periodically traverse the Web in search of WEBSEMANTICS documents and extract the catalog entry. The format of the data maintained by a catalog server is a relation with the following attributes:

$$Catalog[URL, type, translator, source]$$

where URL is the address of the WEBSEMANTICS document describing the data source, *type* is the name of an object type made available by that source, *translator* specifies the address of an interface program that reads data from the database and delivers it using the WEBSEMANTICS protocol (see Sect.7) and *source* is the address of the data source itself. The triple $(typeT, translatorR, sourceS)$ expresses the fact that objects of the type T can be extracted from the data source S using the translator R.

A catalog supports the following functionality.

- If we are interested in finding all WEBSEMANTICS documents that serve as entry points for databases containing a certain *type*, or using a certain *translator*, or accessing a specific *source*, a lookup in the catalog will select the qualifying URLs.
- On the other hand, if we have the URL of a potential WEBSEMANTICS document, and we want to find out if we can extract a certain type, the server will look it up in the table, and if it can find it will return a triple with the corresponding connection information; if the URL is not found in the table, then the server will fetch the document referred to by it, scan it for `CatalogEntry` triples and add them to the table.

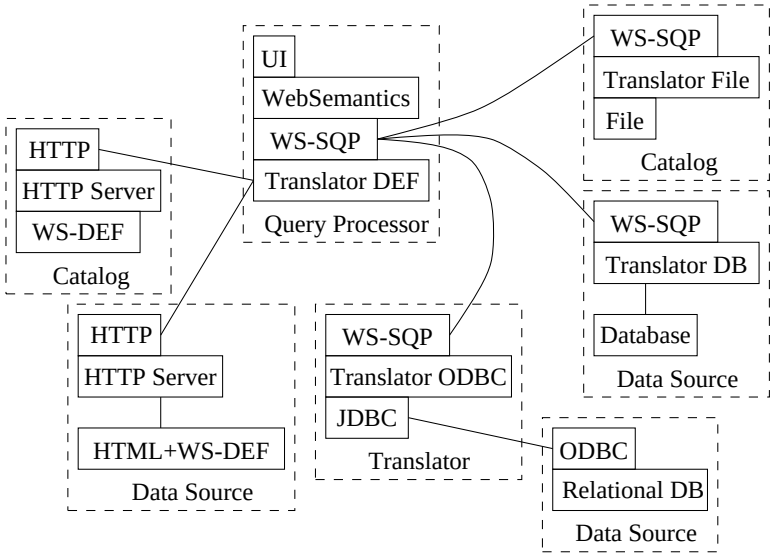

Figure 4: An example configuration of WEBSEMANTICS components. Dotted boxes refer to hosts. Small boxes indicate either protocols or software subsystems. Lines indicate the exchange of queries and answers.

4 Architecture

In this section we describe the architecture of WEBSEMANTICS. Our description is based on an example configuration of components shown in Figure 4. Each large dotted box represents a component. A component has a separate address space and resides on a separate host. Lines between components represent the exchange of queries and answers, in various protocols. A *data source* component has stored data — in this case a collection of HTML documents, or a relational database engine and a database (of some sort). The type (schema) of each data source is different. A *catalog* component also has stored data — but this data is the location, type and translator information needed to access a data source or another catalog. There are multiple independent catalogs in the system. The type of all catalogs is the `catalog` type provided by WEBSEMANTICS. Catalog components are similar to data source components; the only difference is that catalogs have a fixed type and semantics associated with them. A *translator* component provides translation from some protocol to the WEBSEMANTICS data exchange protocol. Translators isolate both the work of the translation of a query into a local format and the work of translating answers back. In our example, a translator component translates ODBC. Finally, there is the WEBSEMANTICS *query processor* component that provides a query service to the user.

Each component consists of subcomponents. For the *data source* component, the subcomponents depend on the data source and on the type of translation services provided. In this example, the database data source component contains two subcomponents: a database and a translator. The translator exports the WEBSEMANTICS *sub-query protocol*, indicated by the *WS-SQP* box. The HTML *data source* component contains two subcomponents, entirely dependent on the WWW; they are the HTML documents and the HTTP server. That data source does not need a translator and does not provide any direct support for WEBSEMANTICS. The *query processor* component of WEBSEMANTICS provides support to process WEBSEMANTICS data resident in HTML documents, since it has a translator subcomponent for HTML documents. The *translator* component shown in the diagram contains a single subcomponent: the translator for ODBC. The translator uses two protocols to accomplish this task: the WS-SQP protocol and the ODBC protocol. The *query processor* component contains three subcomponents: *UI*, the user interface, *WebSemantics*, the WEBSEMANTICS query processor, and *Translator DEF*, a translator that understands WS-DEF, the WEBSEMANTICS data exchange format.

Interactions between components occur in two ways: updates to the catalogs and query processing. We envision two kinds of updates to the catalogs – explicit updates by users (via HTML forms based input) and robot-based scanning of HTML files that contain catalog information expressed in WS-DEF.

Query processing proceeds as follows. A query is entered into the user interface (UI). The query is parsed and checked for catalog searches. If catalog searches exist, they are performed and a set of results from the

[3]To see a catalog entry in an HTML document, we plan to provide a Java applet that can visualize it.

catalogs constructed. The remaining portion of the query is processed by the WEBSEMANTICS query processing engine. This engine issues sub-queries to the appropriate translators and combines the results, which are then returned to the user. As mentioned in the introduction, to call a translator, three pieces of information are needed: the location of the data source; the type processed by the translator; and the sub-query to be processed by the translator or the data source. Depending on the translator, the type processed by the translator may be located in the translator itself, or passed as an argument to the translator. Similarly, the location of the data source may be located within the translator, or it may be passed as an argument to the translator.

5 Query Language

In this section we introduce our WEBSEMANTICS query language WSQL whose purpose is to provide declarative location and access to data. For location of data sources the language uses constructs derived from WebSQL [MMM96]. The data manipulation is expressed using standard select-project-join constructs. The precise syntax of WSQL is given in Appendix B.

To illustrate the various features of the language we use our running example with restaurant and hotel information from the introduction.

Suppose we are interested in finding all restaurants in Paris meeting certain criteria but we don't know where this information is located. All we know is that there is a registered type `Restaurant` used by people exporting this information. Assuming that the schema is available (by consulting a catalog server), we can write a query like the following:

```
select r.name, r.phone
from Restaurant r
where r.city = "Paris" and r.price < 200;
```

This query is evaluated by consulting one or more catalog servers[4], in search for data sources that contain `Restaurant` objects and then querying each of these sources. This type of query presents the highest degree of transparency for the location of data since it doesn't require the user to have any a priori information about the repositories containing the data. The downside of this is, of course, performance, because a potentially large number of data sources is contacted out of which many may not be relevant to the query.

To make up for this inconvenience, we provide the user with the ability of specifying a data source, when its location is known, like in the query below:

```
select r.name, r.phone
from Restaurant r in "http://www.michelin.fr/restaurants.html"
where r.city = "Paris" and r.price < 200;
```

Here we assume that the specified (fictive) HTML document is a WEBSEMANTICS catalog entry (see Section 3) for the Michelin guide online. Transparency in data access is expressed through the `in` keyword, which provides the link between the description of the location of data and the data itself. The `from` clause defines the range of the variable `r` as the collection of all the `Restaurant` objects that are accessible from the specified WEBSEMANTICS document.

Suppose now that a server hosts several WEBSEMANTICS documents, each pointing to different databases and that we are interested in data found in all these databases. Furthermore, assume that all these documents are accessible via hypertext links from the server's root page. The first step towards accessing the actual data is finding these documents. This can be accomplished by using *path regular expressions* a la WebSQL:

```
select r.name
from Document d such that "http://server.com" ->* d,
     Restaurant r in d;
```

Out of all the documents hosted by the server mentioned in the above query, only some are WEBSEMANTICS documents, the others being regular HTML documents. The regular documents will be skipped in the evaluation of this query.

Another method for finding WEBSEMANTICS documents is based on keyword search. This can be done by using a `mentions` clause:

[4]The list of catalog servers used is specified at startup.

```
select r.name
from Document d such that d mentions "Paris restaurant",
     Restaurant r in d;
```

The first `from` sentence restricts the range of documents to those that mention the phrase "Paris restaurant". Each such document is then examined and if it is a WEBSEMANTICS document exporting `Restaurant` objects, all those objects are extracted.

The functionality of the `mentions` predicate is very convenient. However, it accesses a broad number of documents, and the contents of the documents are not known to the user. To restrict the search to relevant documents, we need a list of all the sites containing a specific data type, the sites using a specific translator, or the sites referencing a particular database. This can be accomplished by sending a query to a *catalog server* (described in Sect. 3). For example, suppose we know the address of a data source containing data of interest, but we don't know all the connection information needed to contact that source. We can rely on the system to extract this information from the catalog server, connect the data source and extract the required data, by submitting a query like the following:

```
select r.name
from Document d such that d exports source "db.server.com",
     Restaurant r in d;
```

Thus, the `exports source` keyword instructs the query processor to contact the catalog and extract catalog entries containing a particular source address. The domain of WEBSEMANTICS documents can also be defined through an `exports type` keyword, which matches a particular type, or an `exports translator` construct which matches a particular translator.

Finally, the WEBSEMANTICS data model supports relationships. To use relationships, the WEBSEMANTICS query language borrows constructs from OQL [C+96] for path traversal. For example, to retrieve the address of the business office for restaurants we use a path expression:

```
select r.name, r.headquarters.address
from Restaurant r;
```

6 Query Processing

6.1 Virtual Algebraic Machine

To answer a query the WEBSEMANTICS query processor parses it and generates a tree of algebraic operations. The tree represents the algorithm for processing the query. The run-time system uses the Graefe iterator model (presented in [Gra93]). In order to model the various constructs in the language we introduce several algebraic operators:

6.1.1 Operators for the Location Phase

- *mentions*: $String \rightarrow List(URL)$ outputs a list of URLs whose associated documents contain the string *String* in their text;
- *traverse*: $path \times List(URL) \rightarrow List(URL)$ reads a list of URLs a and returns a list all the URLs accessible from a by following all paths from each URL in a that match the path regular expression *path*;
- *export*: $feature \times value \rightarrow List(URL)$ outputs a list of all URLs of WEBSEMANTICS documents that contain a catalog entry with the $feature$ equal to *value*. The parameter $feature$ can be `type`, `translator` or `source`;

6.1.2 Operators for Transparent Access

- *deref*: $type \times List(URL) \rightarrow List(dataSourceSpec)$ extracts all the catalog entires with type *type* from the WEBSEMANTICS documents associated with the URLs in the input;
- *extract*: $query \times type \times List(dataSource) \rightarrow List(dataObject)$ sends *query* to the translators specified in *dataSource* and outputs the union of data objects of type *type* that the translator returns; the query may only contain selections and projections;

6.1.3 Operators for Data Manipulation

- *singleton*: $URL \rightarrow List(URL)$ constructs a singletone output list of its URL input parameter;
- σ_P: $List(dataObject) \rightarrow List(dataObject)$ is the regular select operator from relational algebra where P is a predicate;
- π_{map}: $List(dataObject) \rightarrow List(dataObject)$ is the regular project operator from the relational algebra where *map* is a mapping of attributes;
- $\bowtie_P$: $List(dataObject) \rightarrow List(dataObject)$ is the regular join operator from the relational algebra where P is a join predicate;

6.2 Query Processing

The query parser generates a query execution tree whose nodes are labeled with operators from the set described in the previous section. Consider for example the following query:

```
select r.name, h.name
from Document d such that "http://www.michelin.fr" ->* d,
     Restaurant r in d,
     Document g such that g mentions "hotel Paris",
     Hotel h in g
where r.arrondissement = 6 and h.arrondissement = 6
  and r.stars >= 1 and h.stars >= 4
  and r.street = h.street;
```

This query can be translated in the tree shown in Figure 5. In the figure we use a slightly different notation for operators to clearly indicate which arguments are the result of data processing and which arguments are the result of the construct of the tree.

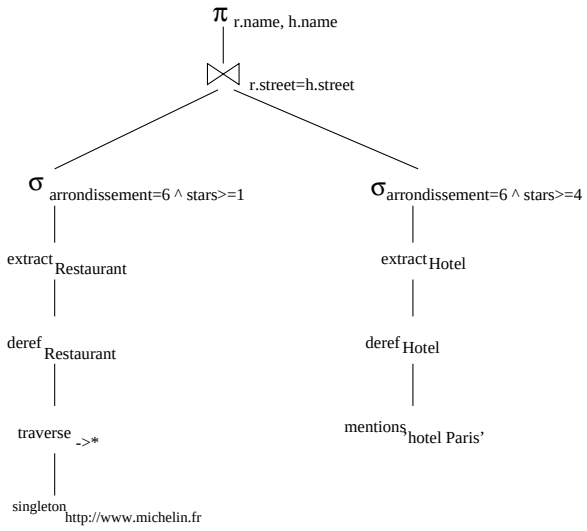

Figure 5: An unoptimized operator tree.

According to this tree, the query is evaluated as follows: first, the $traverse_{\rightarrow *}$ operator finds all the documents reachable from the Michelin home page[5]; then, for each WEBSEMANTICS document in the collection $deref_{Restaurant}$, the operator will extract all the specifications of data source containing restaurants; then, the $extract_{all}$ operator will extract all `Restaurant` objects from each data source; a similar processing is done in the right subtree for `Hotel` objects; the remaining of the tree consists of regular relational operations that compute the final answer.

[5]Some HTML pages will contain catalog entries; other regular HTML pages will simply be ignored.

6.2.1 Optimization

Currently our system performs only a few optimizations for query processing. Cleary many possible optimizations are possible. In this section we give an example of one such optimization. Note that the optimization depends on the ability of translators to execute sub-queries.

One obvious problem with the above execution strategy is that all data objects are extracted from the data sources, which can seriously affect the performance. To avoid this unnecessary data transfer, parts of the query can be shipped to the corresponding translators associated with the data sources, shown in Figure 6.

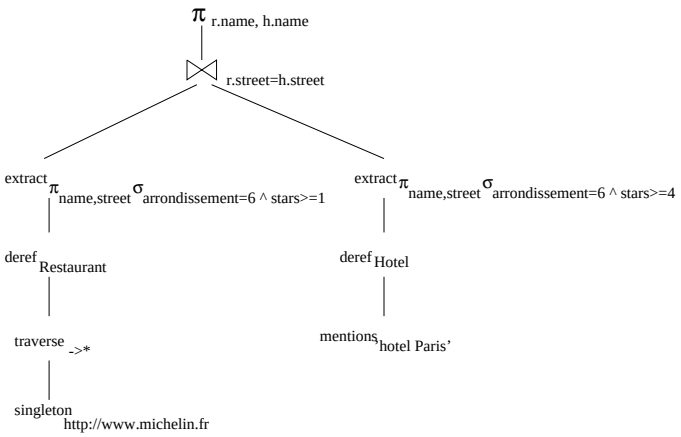

Figure 6: Shipping sub-queries

The following algorithm produces an operator tree for a query. This algorithm processes the atoms in the `from` clause, generating the corresponding sub-trees modeling the computation of the domain associated with each variable. The array *Operator*[] stores in the slot associated with a variable x a handle to the operator that produces the values for that variable. This array is then consulted in order to build the links between the operator nodes.

Step 1:
for each atom A in the `from` clause
switch(A)
case "`Document` d `such that` d `mentions` *keyword*":
Operator[d] = new mentions(*keyword*)
case "`Document` d `such that` d `exports` *feature value*":
Operator[d] = new exports(*feature*, *value*)
case "`Document` d `such that` u *pathRegExp* d":
Operator[d] = new traverse(*pathRegExp*)
Operator[d].input = Operator(u).output
case "*type* x `in` d":
op = new deref(*type*)
op.input = Operator[d].output
Operator[x] = new extract(null)
Operator[x].input = op.output
end switch
end for

Step 2:
for each data variable x
collect all `where` predicates of the form x.*pathexp op constant*
into a sub-query q and place it as argument in the corresponding
extract operator
end for

Step 3:

Build the rest of the tree using standard techniques from relational databases, using Operator[x] instead of the base relation for each data variable x.

Finally, we note that the query processor does not isolation properties (in the sense of transaction processing) of queries. For the database aspects of our system, there is much research in transaction consistency in heterogeneous databases which we have yet to consider. For the information retrieval aspects of our system, to our knowledge, little work has been done on establishing isolation properties over systems such as WWW.

7 Translators

In this section, we discuss several aspects of translators. We discuss three classes of translators, the translator interface for sub-queries and answers, and a special case translator that directly extracts objects in WS-DEF format. Finally, we discuss the implications of the lack of a global object identity for the translator interface and for the WEBSEMANTICS system as a whole.

7.1 Classes of Translators

To function properly, all translators need a data source address and a type. We classify translators by the location of this information. The first class of translator has the data source location and types built directly into the translator. This case corresponds to *wrappers* in heterogeneous database systems. Thus the translator always accesses data from the same data source. The second class of translators provides a gateway type of access with respect to another protocol, e.g., JDBC. This class of translator requires a data source location to be passed as an argument to the call of the translator. The translator then dynamically extracts the types from the data source. The third class of translator requires both the data source location and type to be passed as arguments. It too encodes access with respect to a certain object interchange protocol. The type is used both to type-check the sub-query and to extract data from the data source. All translators support the WEBSEMANTICS sub-query protocol, described next.

7.2 Sub-query Protocol (WS-SQP)

A translator interacts with the sub-query protocol in two phases. First the translator registers itself with a name server, then it accepts sub-queries and returns answers to them. In our implementation, based on DISCO, the Java Remote Method Invocation system is used. Thus a name server has an address, for example, `rmi://opera.db.toronto.edu`. A translator registers itself with the name service under a given name, say `TranslatorServer`. The URL to contact the translator is the concatenation of the two strings, i.e.,

```
rmi://opera.db.toronto.edu/TranslatorServer
```

The translator program then waits, ready to service sub-queries. There are three phases for servicing a sub-query from a translator; the connect phase, the query phase and the data access phase. During the connect phase, a URL of a particular translator is used to obtain an instance of the `TranslatorServer`, say `ts1`. During the query phase, a collection of arguments are passed to the object `ts1` which returns a query object, `q1`. The arguments that are required depend on the class of translator that has been invoked. In any case, an object `q1` results. This object accepts sub-queries and generates answer objects, say `a1`. The object `a1` has three methods in accordance with a classical iterator model. These methods are `open`, `getNext`, and `close`. These methods are invoked remotely by the WEBSEMANTICS query runtime system to access data. The `open` method initiates query processing. The `getNext` method returns each element in the answer to the sub-query. The `close` finishes the sub-query and performs any needed clean up operations.

7.3 Example of Translator for HTML Documents

WEBSEMANTICS has a special translator provided directly in the run-time system to access objects in DEF format embedded in HTML documents. The encoding of the WEBSEMANTICS objects is described in Section 2.

This translator is invoked by the $extract_{query}$ operator. When this operator is executed, the data source is inspected to see if it is an HTML document. In that case, the special translator is invoked to execute the sub-query *query*. The sub-query is restricted to use only select, project, and scan operations.

```
select x.name, x.headquarters.name
from Restaurant x
where x.name = "FourSeasons"
```

```
select x
from Restaurant x
where x.name = "FourSeasons"
```

Figure 7: Two different possible sub-queries for translators.

To construct a query object from this translator, the argument the the URL of the HTML document is passed and optionally a type. Since the HTML document contains an encoding of the type, the type information can be obtained from the document in the case that the type is missing. The result is a query object that can accept queries on the document. Given a query, the resulting answer object supports the same methods `open`, `getNext`, and `close` for data access so that it can be smoothly integrated into query processing.

7.4 Compatibility with Generic Translators

For a more complicated example of a translator, we consider a source which is an Oracle database, which supports ODBC. We envision that the translator for such a source will be implemented using the JDBC interface to ODBC. The WEBSEMANTICS run-time system would invoke an instance of the JDBC translator; using the RMI protocol, as described previously. The translator accepts a sub-query and converts it into an SQL query for the Oracle ODBC server. Tuples are accepted from the server and the translator would convert ODBC tuples into WEBSEMANTICS objects. This translator instance would support the method `open`, `getNext`, and `close` in the usual way.

7.5 Use of OIDs and Effect on Translators

Supporting object identity (OID) and object reference (ODMG relationships) is an open issue in heterogeneous database environments based on the object model. A trivial solution is to not include OIDs in the model. All accesses to the translators must be value based. If OIDs are indeed required in the model, then a global OID can be supported across all sources, or a local OID may be made visible across the translator interface. Mandating a global OID is often not feasible. Support of a visible local OID has been supported in systems such as IRO-DB [G+96] and Garlic [C+95], but each of these solutions has had to resolve complex issues. The option that we have chosen in WEBSEMANTICS is to allow the model to have OIDs, but not permit OIDs from sources to be made visible across the translator interface.

Given this situation, the answer to a query can be either permitted to be an object or it can be restricted to be value-based (scalars). In WEBSEMANTICS we have chosen that the answer to a query can be an object. Now, this object, and all other objects to which it makes reference must be materialized across the translator interface. The WEBSEMANTICS data exchange format that was discussed previously in this section is able to handle such materialized objects. Materialization of objects has several implications. One is that there can be no cycles of object references. The second is that either the translator must accept queries that include path expressions, or if the translator will not accept path expressions, then the path expression must be evaluated in the WEBSEMANTICS runtime system.

Consider the following query, where `x.headquarters` is a `Business` object.

```
select x.name, x.headquarters.name
from Restaurant x in "http://server.com"
where x.name = "FourSeasons"
```

This query is processed by the run-time by constructing a sub-query to submit to a translator. A translator that accepted path expressions would accept the *query* on the left in Figure 7 and it would produce scalar values. A translator that did not accept path expressions would accept the *query* on the right and would produce `Restaurant` objects. Another query would have to executed on these `Restaurant` objects, within the run-time system, to produce the answers to the query.

8 Related Work

Current research in heterogeneous databases has attacked in various ways the problem of describing, integrating or accessing structured data on the INTERNET. However, little research has been done on the problem of *locating*

data sources. This is the main contribution of WEBSEMANTICS. However, WEBSEMANTICS is not a stand-alone technology, and it depends on the existence of other technologies. In this section, we review this research. We consider industry-wide open database standards and wrapper mediator architectures; both are critical for WEBSEMANTICS. We also review some alternate technology such as Harvest/Essence [BDH+95]; InfoSleuth [B+97]; and WWW query languages [KS95, MMM96]. In addition, there has been important research in access to semi-structured databases [AQM+96, BDHS96]; ontologies [Gru93]; and in semantic heterogeneity and information integration in databases [Ken89, Kim95].

There are currently two major competing standards for database integration: CORBA (the Common Object Request Broker Architecture), proposed by OMG (Object Management Group) [Gro92], and OLE/DB (Object Link Embedding for Data Bases), by Microsoft. Since 1989, OMG has been concerned with the definition of an open object infrastructure, or "global bus for distributed components", known as CORBA. One of the defining aspects of CORBA is that it creates *interface specifications* instead of code. The specifications are written in a neutral *Interface Definition Language (IDL)*. The IDL ensures the portability of components and data across languages, tools, operating systems and networks.

At Microsoft, efforts have been made towards the development of a set of interfaces whose goal is to enable applications to have uniform access to data stored in both DBMS and non-DBMS repositories. The main idea is to benefit from database technology, without transferring the data into a DBMS. The approach consists in defining an open, extensible collection of interfaces that encapsulate orthogonal portions of DBMS technology. This set of interfaces, collectively referred to as OLE/DB, is described in [Bla96]. One key idea of this approach is the concept of component DBMSs, which consist of several independent data providing units, communicating with the main query processor through a set of well defined interfaces. This facilitates data integration while imposing minimal requirements on the functionality of the data sources. Unlike ODBC (Open Database Connectivity standard), which requires data providers to implement SQL access to data, the OLE/DB standard can integrate sources that implement interfaces for tabular access, at the minimum.

We now consider wrapper mediator architectures, as proposed in [ACPS96, B+97, C+95, G+96, KLSS95, R+89, P+96, TRV96, Wie92]. These systems differ widely in the capabilities of mediators and in the capabilities of wrappers. However, we believe that interoperation between WEBSEMANTICS and these systems is possible.

WEBSEMANTICS differs in two distinct ways from these systems. First, all of the above systems assume that the location of the data sources and the types are embedded within the wrappers (translators). We lift this restriction in our system. Second, all of the above systems do not provide facilities for the location of components such as mediators and wrappers. This facility is a central contribution of this paper.

The importance of the World Wide Web as a repository of information has generated an increasing interest in the research community for the design of high-level, declarative languages for querying it. WebSQL [MMM96] integrates textual retrieval with structure and topology-based queries. Instead of trying to model document structure with some kind of object-oriented schema, as in [CACS94, QRS+95], a minimalist relational approach is taken: each Web document is associated with a tuple in a virtual `Document` relation and each hypertext link with a tuple in a virtual `Anchor` relation. In order to query these virtual tables, one must first define computable sub-domains, either using keyword matching or through controlled navigation starting from known URLs. Another Web query language, W3QS [KS95] includes the specification of syntax and semantics of a SQL-like query language (W3QL) that provides simple access to external Unix programs, advanced display facilities for gathered information, and view maintenance facilities.

Distribute information retrieval systems, for example the Harvest/Essence information retrieval based system [BDH+95] are indirectly related to our work. Essence is a customizable information extraction system that is used to extract and structure mostly textual infromation from documents in Harvest. It exploits the formats of common file types and extracts contents of files. The result is a summary object (a SOIF record). Collections of SOIF records are are indexed and organized into *brokers*, a kind of mediator. Brokers provide information retrieval search on their associated SOIF records. SOIF records are related to the WS-DEF object format. However, these systems focus passive documents and information retrieval search whereas we focus on querying strictly typed data in structured and semi-structured data sources. Another systems [WID97] focus on CORBA IDL object access via WWW protocols instead of query based access.

InfoSleuth [B+97] proposes information brokering and domain ontologies as two ways to handle data sources. We believe that domain ontologies are promising extension to the catalog server, since they provide a way to structure domain information (types). However, this system does not propose specific techniques for identifying types and translators.

9 Conclusion

9.1 Summary of Results

We have presented a new system, WEBSEMANTICS, that gives "equal-time" to data access on the INTERNET. We described our data model and we introduced an encoding of typed data in HTML documents to facilitate light-weight publishing of data. We proposed an architecture based on components that can be integrated in arbitrary ways, to support specific applications. Thus, we introduced four types of components: *data source* components, which contain data stored in a collection of HTML documents or in a database; *translator* components, which provide translation from some protocol to the WEBSEMANTICS data exchange protocol; *catalog* components, that contain the type, translator, and location information needed to access a data source or another catalog; and *query processing* components that provide query processing capabilities, in an integrated way, over the other three components.

To allow for dynamic location of data sources we proposed a special type of HTML document pairing data source connection information with a textual description of the data source content. These documents enable us first to use information retrieval techniques for the location of relevant data sources and second to permit the easy publishing of data source connection information.

Furthermore, we introduced our WEBSEMANTICS query language (WSQL) whose purpose is to allow declarative location, access and manipulation of data. To support the constructs of the query language we introduced an appropriate set of algebraic operators and we gave an algorithm that constructs a query evaluation tree.

We described the interface and the operation of various types of translators and catalogs in conjunction with the query evaluation engine. The overall result is, we believe, a system that provides an alternative to the WWW for access to structured data. Given a large population of components and an agreed upon semantics for data, a world-wide body of users can easily exchange data.

9.2 Implementation Status

In this section we describe the status of our prototype implementation of the WEBSEMANTICS system.

The WEBSEMANTICS prototype has four main components: the WSQL Compiler, the Query Engine, a Catalog Server and the Translators. All these components are implemented as a collection of Java [SM] classes, organized in a WEBSEMANTICS Class Library, which facilitates their integration in Java application programs.

WSQL Compiler. The WSQL compiler parses the query and, if no errors are present, translates it into a program in a custom-designed abstract machine used for query execution.

Query Execution Engine. The program is executed by an interpreter that implements a stack machine. Its stack is heterogeneous, that is, it is able to store any type of object, from integers and strings, to whole vectors of WEBSEMANTICS data objects. The evaluation of the ranges specified in the `from` clause is done via designated operation codes implementing the algebraic operators described in Section 6.1.1 and Section 6.1.2. Whenever the interpreter encounters an operation code corresponding to a range specifying condition, the query engine is invoked to perform the actual evaluation. Depending on the operation, this involves sending a request to an index server (for the *mentions* operator), to a catalog server (for the *exports* operator), or a depth-first traversal of a sub-part of the document network (for the *traverse* operator). After the document variables have been instantiated, the appropriate translators are contacted to extract the data from the data sources described in each document.

Catalog Server. For the first prototype, we have implemented a simple catalog storing connection information for a small collection of data sources. The catalog is accessible through an RMI interface containing the `open()`, `get_next()` and `close()` methods. The `open()` method is invoked with a $(feature, value)$ pair as argument (where $feature \in$ {`url`, `type`, `translator`, `source`}.), thus specifying the selection condition $feature = value$ to be applied to the catalog's table. The tuples containing connection information are retrieved through successive `get_next()` calls. Finally, a call to `close()` closes the data stream.

Translators. So far we have implemented only two translators, named `FileTranslator` and `UrlTranslator`. Both of them are able to extract data objects from WEBSEMANTICS HTML documents using the encoding described in Sect. 2. The only difference between the two translators consists in the access method used for reading the HTML files: the former accesses local files whereas the latter retrieves files from the Web using the HTTP protocol. Neither of the translators supports select-project queries for now[6], and therefore are only able to extract *all* objects of a given type from a source.

[6]Fully operational translators are under development.

The prototype uses a catalog at URL `rmi://opera.db.toronto.edu/CatalogServer`. The fully functional WEBSEMANTICS prototype, together with a comprehensive list of WSQL query examples is accessible at

`http://www.cs.toronto.edu/~georgem/WebSemantics`

Acknowledgments

Thanks to Helenas Galhardas, Françoise Fabret and Philippe Monneret for comments on earlier drafts of this paper.

References

[ACPS96] S. Adali, K. S. Candan, Y. Papakonstinou, and V. S. Subrahmaniam. Query caching and optimization in distributed mediator systems. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 137–148, 1996.

[AQM+96] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. Technical report, Department of Computer Science, Stanford University, 1996. http://db.stanford.edu/pub/papers/lorel96.ps.

[B+97] B. Bohrer et al. InfoSleuth: Semantic integration of information in open and dynamic environments. *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1997. To Appear.

[BDH+95] C. Bowman, P. Danzig, D. Hardy, U. Manber, and M. Schwartz. The Harvest information discovery and access system. *Computer Networks and ISDN Systems*, 28:119–125, 1995.

[BDHS96] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1996.

[Bla96] J. Blakeley. Data access for the masses through ole db. *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1996.

[BRU97] P. Buneman, L. Raschid, and J. Ullman. Mediator languages – a proposal for a standard. *ACM SIGMOD Record*, March 1997. To Appear.

[C+95] M. Carey et al. Towards heterogeneous multimedia information systems: the Garlic approach. Technical report, IBM Almaden Research, 1995.

[C+96] R.G.G. Cattell et al. *The Object Database Standard - ODMG 93, Release 1.2.* Morgan Kaufmann, 1996.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 313–324, 1994.

[G+96] G. Gardarin et al. Iro-db: A distributed system federating object and relational databases. In O.A. Bukhres and A.K. Elmagarmid, editors, *Object-Oriented Multidatabase Systems : A solution for Advanced Applications.* Prentice Hall, 1996.

[Gra93] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2), June 1993.

[Gro92] The Object Management Group. *The Common Object Request Broker: Architecture and Specification.* QED Publishing Group, Wellesley MA, 1992. OMG Document Number 91.12.1, revision 1.1 edition.

[Gru93] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 1993.

[Ken89] William Kent. The many forms of a single fact. In *Proceedings of Thirty-Fourth IEEE Computer Society International Conference (COMPCON)*, San Francisco, (Spring) February 1989. IEEE.

[Kim95] Won Kim. *Modern Database Systems: The Object Model, Interoperability, and Beyond.* ACM Press, New York, NY, 1995.

[KLSS95] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Proceedings of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, Stanford, California, March 1995. `http://portal.research.bell-labs.com/orgs/ssr/people/levy/im95.ps.Z`.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A query system for the World Wide Web. In *Proceedings of the Twenty First International Conference on Very Large Data Bases*, pages 54–65, 1995.

[LGP97] H. Garcia-Molina L. Gravano, C.-C. Chang and A. Paepcke. STARTS: Stanford proposal for internet meta-searching. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 1997. To Appear.

[MMM96] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. In *Proceedings of Parallel and Distributed Information Systems (PDIS)*, pages 80–91, 1996.

[P+96] Y. Papakonstantinou et al. Capabilities-based query rewriting in mediator systems. In *Proceedings of the International Conference on Parallel and Distributed Information Systems (PDIS)*, 1996.

[QRS+95] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Deductive and Object-Oriented Databases, Proceedings of the DOOD '95 Conference*, pages 319–344, Singapore, 1995. Springer Verlag.

[R+89] M. Rusinkiewicz et al. Query processing in a heterogeneous multidatabase environment. In *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, 1989.

[San95] Sandre. Dictionnaire de données sur l'eau. Technical report, Office International de l'Eau, France, 1995.

[SM] Sun Microsystems. Java (tm): Programming for the internet. `http://java.sun.com`.

[TRV96] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of disco. In *Proceeding of the International Conference on Distributed Computing Systems (ICDCS)*, 1996.

[TS97] A. Tomasic and E. Simon. Improving access to environmental data using context information. *ACM SIGMOD Record*, 1997. To Appear.

[USM96] The USMARC home page. Technical report, Library of Congress, 1996. `http://lcweb.loc.gov/marc/`.

[WID97] Web interface definition language (WIDL), 1997. `http://www.webMethods.com`.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, March 1992.

[WMR97] Joint workshop on metadata registries, July 1997. University of California, Berkeley.

A The BNF Specification of the Data Model

The definition is based on the ODMG schema definition, using ODL, in [C+96]. A user defined type is specified by its interface. The interface definition is the following:

InterfaceDcl ::= `interface` Identifier [InterfaceBody]
InterfaceBody ::= Export | Export InterfaceBody
Export ::= AttribDcl ; | RelationDcl ;
AttribDcl ::= `attribute` DomainType Identifier
DomainType ::= BaseTypeSpec | CollSpec Identifier

RelationDcl ::= `relationship` Target Identifier
Target ::= Identifier | CollSpec Identifier
CollSpec ::= `Set<`Identifier`>` | `Bag<`Identifier`>`

Base types are the atomic/literal types of ODMG. We support (parameterized) collections `Set<t>` and `Bag<t>` where `t` is a base type or a user defined type.

B The BNF Specification of the WebSemanticsQuery Language

Query := `select` AttrList `from` DomainSpec [`where` Condition] ;
AttrList := Attribute {, Attribute}
Attribute := Var.Field{.Field}
Field := Identifier
Var := Identifier
DomainSpec := DomainTerm {, DomainTerm}
DomainTerm := Type Var `such that` DomainCond
| Type Var [`in` SourceSpec]
DomainCond := Node PathRegExp Var
| Var `mentions` StringConstant
| Var `exports type` Type
| Var `exports translator` Translator
| Var `exports source` Source
SourceSpec := Node
| `(` Type, Translator, Source `)`
Type := Identifier
Translator := StringConstant
Source := StringConstant
Node := StringConstant
| Var
Condition := BoolFactor {`or` BoolFactor}
BoolFactor := BoolTerm {`and` BoolTerm}
BoolTerm := Attribute = Attribute
| Attribute = StringConstant
| Attribute `contains` StringRegExp
| (Condition)
PathRegExp := Link
| PathRegExp *
| PathRegExp PathRegExp
| PathRegExp "|" PathRegExp
| (PathRegExp)
Link := `=` | `#>` | `=>` | `->` | Identifier

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399